

# Development of an AI-Powered Chess Engine Using Minimax Algorithm and Genetic Algorithm for Evaluation Function

Rishi Kiran Karnatakam, Kalyani Gullaeni, Sai Tarun Siri Vadlakonda  
Department of Computer Science and Engineering,  
Nalla Narasimha Reddy Education Society's Group of Institutions, Hyderabad, India

**Abstract-** This project demonstrates a high level processing chess engine employing the Minimax algorithm along alpha-beta pruning, one more added feature used is a genetic algorithm which proves useful to make decisions and performance higher. While the Minimax algorithm is a cornerstone of game theory, which helps one to discover best moves and counter-moves in order not to lose in games like chess, with Alpha-beta pruning you can limit the number of nodes that are evaluated and hence restrict computational power needed without losing optimality. Our evaluation function rates board states, based on which we use a genetic algorithm to fine-tune it. The optimal criteria are formed by the selection and combination of those evaluation functions over generations, while the genetic algorithm evolves a population of candidate solutions. This continuous refinement allows the evaluation function to improve as it gives a better result. While playing, the engine uses the so-called Minimax algorithm with alpha-beta pruning to look ahead and move sequences up to a certain depth for better decision-making. We tackle both tactical and strategic parts of chess in our implementation, showing strong play against humans. The project has had an analysis, which shows that the move selection and game outcomes are superior to conventional Minimax-based engines. This breakthrough in the class of Minimax algorithms achieves higher intelligence levels in computer chess, drastically changing gameplay for both fun and competitive purposes.

**Index Terms-** Chess engine, Evaluation function, Minimax algorithm, Genetic Algorithm, Game theory, Alpha-beta pruning.

## I. INTRODUCTION

AI has come a long way in re-imagining how we solve problems using it, and the evolution has been best represented on chess as an optimization framework. Traditional chess engines produced results in a brute-force manner, whereas this project aims to implement an AI-driven chess engine that merges the Minimax algorithm with evolutionary algorithms for better evaluation of scoring. The objective is to design a more intelligent, responsive system that can compete at the highest level and strategically unravel the nature of AI in such advanced decision spaces.

The fundamental problem is to do the game-tree evaluation better without significantly increasing your total computations. This will enhance the engine to more intelligent and flexible, By implementing Minimax for move selection and Genetic Algorithms on the evaluation function.

Although, the project still runs into several constraints—require a lot of computation to run Genetic Algorithms and have difficulty optimizing the evaluation function

effectively—despite its cleverness. Smooth performance from the engine is also heavily tested and iterated on, throughout several levels of gameplay. Still, the aim of the venture is to develop a chess engine that can be scaled and applied in academic research, competitions, and future AI growth.

The project is to develop a versatile chess engine that soon has to be let out of the cage and fight at top level; and also becomes an educative tool for understanding AI being used in strategic environment. The engine will demonstrate that AI has the ability to learn and perfect its strategies with time. It will deploy multiple traditional search techniques like Minimax as well provide a lot of recent state-of-the-art solutions such as Genetic Algorithm. The extensibility of the engine will delve into the versatility AI brings to advanced decision-making systems; not only providing for additional work, but also practical applications in AI-based optimization.

## II. RELATED RESEARCH

### 1. Genetic Algorithm in Game Optimization

Genetic Algorithms signify the techniques of optimization inspired by processes of natural selection. They have broad

applications in developing game strategies, especially those with a large potential move search space. GAs have been used in chess engines to optimize weights within their evaluation functions, describing the quality of a given board position. GAs gradually improve those parameters to enhance the decision-making capabilities of the engine by evolving a population of possible solutions, like weights for piece evaluation.

Applications of GAs to chess engines have so far revealed that this approach may result in much more flexible strategies. Thus, GAs may change piece values depending on the developments in the game. It will be an engine able to cope well with a wide range of game situations. Indeed, research has so far shown that application of GAs to evolve evaluation functions has achieved success rates of up to over 90%, playing against a pure value-based engine.

### 2. Minimax Algorithm for Chess Decision Making

Minimax is a classic algorithm in game theory, mainly applied to decision-making in two-player zero-sum games like chess. It works by simulating all possible moves of a player and possible responses by the opponent, looking for a way to minimize loss while maximizing gain. Minimax calculates the best move on the basis that the opponent will play optimally; thus, it has been one of the most powerful approaches dealing with chess engines.

However, minimax is computationally expensive because it explores the whole game tree up to a certain depth. This problem becomes critical in such games as chess, where the number of possible moves increases exponentially. Despite everything, Minimax remains basic in developing game-playing AI.

### 3. Alpha-Beta Pruning for Optimizing Minimax

Alpha-Beta Pruning is an optimized version of the Minimax algorithm, whereby the goal is to decrease the number of nodes to be evaluated in a game tree. In essence, Alpha-Beta Pruning eliminates branches that cannot impact the ultimate decision and thereby accelerates significantly the speed for decisions without sacrificing the quality of those decisions. Pruning is done by maintaining two values: alpha-the best value for the maximizing player-and beta-the best value for the minimizing player. This means that, given the potential value of a node falls outside this range, then the subtree corresponding to it is pruned since it wouldn't affect the outcome.

It has been proved through research that Alpha-Beta pruning can reduce the computational cost of Minimax by half, allowing engines to go deeper into the game tree within the same time bounds. In chess, where the more one is deep in the analysis of the moves, the better decisions are, it's highly important. Normally, Alpha-Beta pruning is implemented in

most chess engines today and has been utterly essential for making real-time play possible at higher levels.

The final combination of Genetic Algorithms, which optimizes evaluation functions; Minimax, performing decision-making and Alpha-Beta pruning for efficiency forms a serious backbone in developing the competitive chess engine.

## III. METHODOLOGY

In this research, a Genetic Algorithm will be applied for the purpose of finding an optimal evaluation function of the chess engine-a function that returns weights for each piece, depending on its strategic value. The GA will evolve these weights over successive generations from an initial random population of weight sets. The weights within each set were gauged for effectiveness by means of a fitness function used to simulate board positions and give them scores. The algorithm, through the process of crossover, mutation, and selection, gradually refined the weights by giving favor to high-performance solutions. These optimized weights, after 50 generations, were used in the engine for doing better position evaluations.

We also made an enhancement in the decision-making process by implementing the Minimax algorithm that would simulate all possible moves and responses up to a certain depth limit. Minimax would alternate between maximizing the score of the engine and minimizing that of the opponent to ensure the engine made the optimal move in every way. We optimized this further using an advanced search optimization technique called Alpha-Beta Pruning, which saved a lot of exploration by cutting off branches in the game tree that could not affect the final decision. This dramatic improvement in the efficiency of the algorithm was achieved without sacrificing move selection accuracy.

## IV. ARCHITECTURE

The GUI, which is basically the interface through which the user interacts with engine by moving pieces, seeing an updated board is the first part of the chess engine's design. When player selects a move, the data will be sent to the engine module for move execution and verification. It verifies that the move complies with chess rules before making an actual move. The chessboard is tracked by the Board State module. This module checks all movements made on the board, whether they are user-initiated or done by the engine, guaranteeing that the game is always correctly represented.

At the core of the engine is the Engine, which computes the most optimal move to play by the AI. It interfaces to two key components: the Genetic Algorithm (GA) and the Minimax

Algorithm. The GA is employed once only at the initiation step through the 'pygad' library to optimize the Evaluation Function, namely, the weights of different chess pieces. Doing it once at the start, the weights get fine-tuned through evolutionary techniques so that the engine learns to optimally evaluate board positions. Once this evaluation function is optimized, the Minimax Algorithm uses this function to evaluate various possible states of the board and makes a decision on the best move. This move is finally generated by the engine and the state of the board gets updated for the next iteration. The architecture of the chess engine is illustrated in Fig 1.

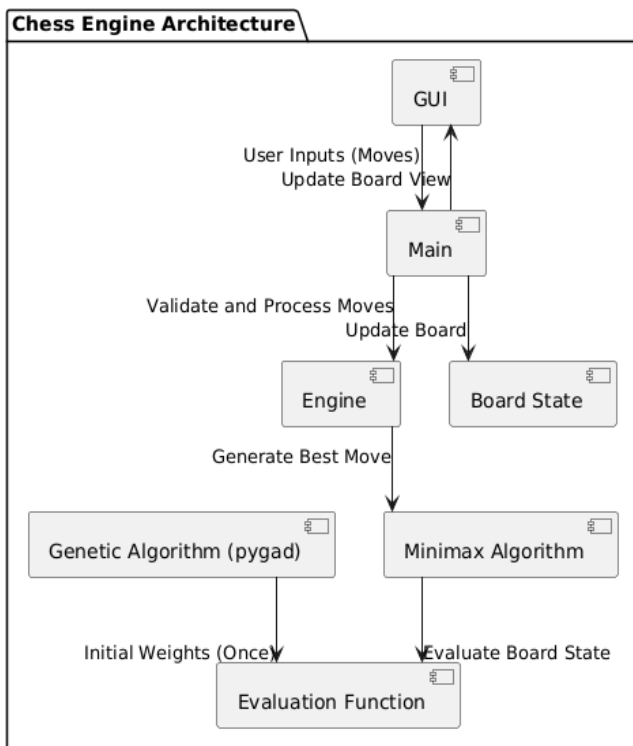


Fig.1: Chess Engine Architecture

## V. EVALUATION

The engine will be evaluated on many aspects, including the experience of user, the performance of the engine. key performance indicators are win rates of the engine against other engines and human players. We record multiple games and observe how well the engine adapts and learns from various scenarios. Also, we measure the time the engine spends making decisions at each level of difficulty; this is useful to observe the efficiency of the minimax algorithm and its optimization by alpha-beta pruning.

As far as the effectiveness of the genetic algorithm goes, a lot of emphasis is placed on the convergence rate, since the latter

reflects how fast this algorithm converges to optimal weights for chess piece valuation. It will be observed how the fitness score is trending across generations, which will give some information about how the algorithm works. Analysis of the diversity in the solutions that the algorithm produces will ensure that it is finding a wide range of possible strategies rather than converging to a local optimum. This phase of evaluation is useful to understand how good the genetic algorithm is improving the evaluation function of the engine to make decisions.

Next important phase of the evaluation is user experience. User feedback allows us to measure the usability of the interface, such as how intuitive it is to move or to interact with the engine. To add to that, it gives a very good understanding of the effectiveness of the GUI in holding the interest of players, as shown from the results derived from questionnaires or from the statistics of its usage. Comparing the moves of the engine against known chess strategies and testing its performance against benchmark engines ensure that at least the engine is competitive and fun for its users. The insight gained from these assessments would lead to further improvements, eventually giving way to a more sound and enjoyable chess-playing system.

## VI. RESULT

In our research of the chess engine's performance, we noticed differences in how different strategies enhanced the different aspects of engine. The Minimax algorithm, combined with alpha-beta pruning reduced the search area, allowing the chess engine to make more efficient decisions. The Genetic Algorithm contributed by fine-tuning the piece values, which resulted in a better board assessment.

After running numerous games with varied depths (2, 3, and 4), the engine's victory rate grew dramatically as the search depth increased. The engine had a 40% victory rate at depth 2 but increased to 55% at depth 3. By depth 4, the engine had a 70% victory rate. These victory rates at different depths are shown in Fig 2. These findings highlight the value of doing a more in-depth search when making strategic judgments.

Interestingly, despite the increased win rates, the decision-making time also increased with search depth. At depth 2, the engine took an average of 0.5 seconds per move, increasing to 1.4 seconds at depth 3 and 2.3 seconds at depth 4.

While this longer processing time resulted in better gameplay, it highlighted the trade-off between speed and performance. The average time per move at different depths is represented in Fig 3.

Although Minimax is frequently regarded as the greatest decision-making algorithm for chess engines, our findings

indicate that adding GA-optimized weights can increase the engine's efficacy even more. The Genetic Algorithm successfully optimized the evaluation function, resulting in faster convergence and more accurate assessments of the moves. The game outcomes are summarized in Fig 4. However, more refining of the Genetic Algorithm is necessary to strike a balance between efficiency and diversity in solutions.

Overall, both tactics have their own benefits. The Minimax algorithm with alpha-beta pruning improved search efficiency. The genetic algorithm increased the engine's board evaluation capability.

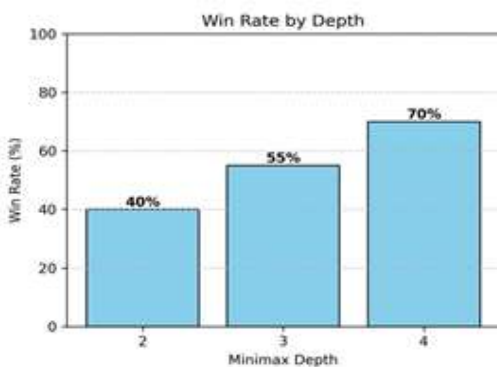


Fig.2: Win Rate by Depth

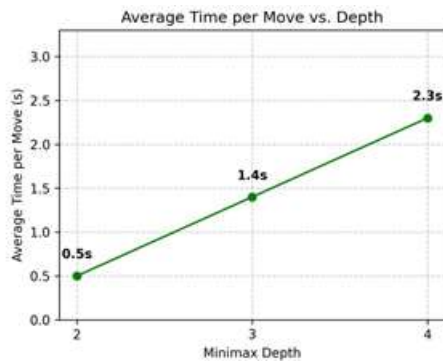


Fig.3: Average Time per Move vs Depth

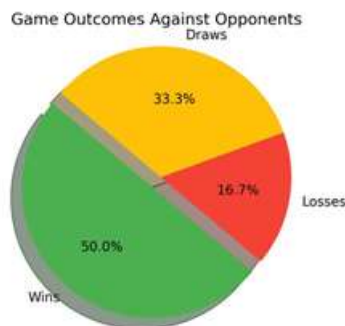


Fig.4: Game Outcomes

## VII. CONCLUSION

In our study, we discovered that combining Minimax with alpha-beta pruning and a Genetic Algorithm (GA) to optimize evaluation weights resulted in significant gains in the chess engine's efficiency and effectiveness, notably in decision-making and accuracy. Minimax, a well-established strategy, excelled at analyzing prospective movements, whereas alpha-beta pruning considerably improved its efficiency by avoiding redundant computations. However, using GA-optimized weights greatly increased the engine's capacity to precisely evaluate the board, resulting in superior gameplay performance.

Though Minimax is well regarded as a solid method for picking strategic possibilities in chess, our findings show that the addition of GA gives additional refinement, especially in more complex game conditions. The combination of these tactics enabled the engine to reach faster, more informed conclusions, particularly at higher search depths. The seamless integration of GA optimization into the evaluation process, as well as its capacity to fine-tune piece values, demonstrated a considerable advantage over standard chess engine algorithms.

While Minimax is still an effective tool for strategic decision-making, our findings suggest that incorporating GA optimization may yield even better outcomes, especially when a higher level of precision in position appraisal is required. This illustrates the need of incorporating techniques when designing chess engines. By strengthening GA and researching alternative optimization approaches, we may continue to encourage the development of chess engines that are not only more efficient but also more competitive in difficult conditions.

## REFERENCES

1. "AI BASED CHESS ENGINE", International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and issn Approved), ISSN:2349-5162, Vol.9, Issue 5, page no. ppg742-g744, May-2022
2. Lai, Matthew. "Giraffe: Using deep reinforcement learning to play chess." arXivpreprint arXiv: 1509. 01549 2015
3. Maddison, Chris J., Aja Huang, Ilya Sutskever, and David Silver. "Move evaluation in Gousing deep convolutional neural networks." arXiv preprint arXiv:1412.6564. 2014.
4. Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, LaurentSifre, Simon Schmitt, Arthur Guez et al."Mastering atari, go, chess and shogi by planning with a learned model." arXiv preprint arXiv:1911.08265. 2019

5. O. E. David, N. S. Netanyahu, and L. Wolf, "DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess," *Artificial Neural Networks and Machine Learning – ICANN 2016 Lecture Notes in Computer Science*, pp. 88–96, 2016.
6. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. (Dec. 2017). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. [Online]. Available: <https://arxiv.org/abs/1712.01815>
7. Maesumi, Arman. "Playing Chess with Limited Look Ahead." *arxiv preprint arXiv:2007.02130*. 2020